111-63
161737
17P.

NASA TECHNICAL TRANSLATION **NASA TT - 20357**

SPOCK--A TOOL FOR REAL TIME ARTIFICIAL INTELLIGENCE

J. P. LaCroix, G. A. Berthon, F. Fages, P. Repusseau

Translation of "Spock, un Outil pour l'Intelligence Artificielle Temps Reel", AGARD, 55th Meeting of the Avionics Panel of Software and Engineering and its Application to Avionics. Held in Cesme, Turkey, April 25-29, 1988

(NASA-TT-20357) SPOCK: A TOOL FOR REAL TIME ARTIFICIAL INTELLIGENCE (NASA) 17 p CSCL 09B

N89-10574

Unclas
G5/63 0161737

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D. C. 20546 AUGUST **1988**

SPOCK - A TOOL FOR REAL TIME ARTIFICIAL INTELLIGENCE

J.P. LACROIX; G.A. BERTHON; F. FAGES; P.REPUSSEAU
Thomson-Csf AVG
31 Rue Camille Desmoulins
92130 Issy-les-Moulineaux France

Thomson-Csf LCR
Domaine de Corbeville
BP 10
91401 Orsay France (F. Fages)

Abstract: Airborne expert systems must take into account very harsh real time conditions which, when combined with the environmental conditions of the object computer, require the development of new solutions. The proceduralization obtained by compiling the RETE algorithm toward symbolic, procedural, or real-time oriented target languages provides a solution to the primary problems of airborne applications: speed requirements, memory control, harmonization of symbolic parts with the traditional portions of the applications. This approach is supported by the SPOCK family of tools (-LISP, -C, -LTR3, -ADA) developed by the Central Research Laboratory of Thomson-Csf which offers significant improvements over presently available products. One of the SPOCK applications in Thomson-Csf is a helicopter mission aid system which takes into consideration the threats and terrain thanks to integration with an airborne cartographic data base.

1 CONDITIONS OF REAL-TIME AIRBORNE EXPERT SYSTEMS

The operational use of expert systems in airborne military applications must take into account the specificities of these applications and the object computers must be suited to the severe environmental conditions encountered.

In the case of the expert system portion the following needs arise:

- Analysis of the situation and real-time decision making in the avionic sense of the term (less than 0.1 second)
- Consideration of the time and space aspects in the reasoning
- Ability to process truncated or incomplete information

- Coexistence/merging with conventional applications or processing (algorithmics/computations/input-output management, etc.)

For the airborne machine the requirements are to offer the following items in the smallest possible volume, with the lowest possible power consumption, and under stressful temperature and vibrations conditions:

- Extended memory capacities
- Very high processing capabilities compared to traditional processing needs.

2 POSSIBLE SOLUTIONS

In order to satisfy the processing needs three different approaches can be considered:

2.1 RAW POWER

In this case the object computer has the greatest possible computing power (a few MIPS or more) in virtue of the equivalency between an inference and a few tens of traditional instructions. The expert system itself is written in a conventional language, such as Pascal or C, or possibly a real time language such as LTR3 or ADA.

ADVANTAGES:

- No expensive mechanism for memory recovery to be developed
- Easy integration with the rest of the application which is also written in a language of this type.

DRAWBACKS:

- Very poor symbolic development environment in the case of the languages mentioned
- Difficulty or complexity in creating certain parts of the expert system
- Doubts as to this approach's ability to create powerful "symbolic" applications.

Example: HEXSCON expert system by SRI.

2.2 PURE SYMBOLIC

2.2.1 On a Traditional Machine

This is an approach based on the preceding machine philosophy with the use of a symbolic type language (COMMON-LISP, PROLOG, etc.) compiled for better performance.

ADVANTAGES:

- The development environment is very rich.
- The performance can be very good under certain conditions (no activated memory recovery).

DRAWBACKS:

- Since these machines do not have any hardware memory recovery mechanisms, they have to stop all processing during variable intervals (depending on the size of the installed memory and the problem being processed) and are of little use to real time applications.
- The software interfaces with the traditional languages are not very practical.

EXAMPLES: SUN development stations (type 3-xxx), TECTRONIX, HP, etc.

2.2.2 Dedicated Architecture Machine

These machines are built around specialized architectures (association of tags with data, hard-wired memory recovery, mechanisms adapted to the specificities of the language, etc.) for symbolic languages.

They can be used for airborne artificial intelligence provided that they are freed of their development environment to be made into object computers.

ADVANTAGES:

- The machine identity of development/object computer ensures the repeatability of the performance characteristics from the laboratory to the application.

DRAWBACKS:

- The same as in paragraph ?. for the software
- The need to reduce the size of the computer for extensive integration actions.

EXAMPLES:

- Compact Lisp Machine by TEXAS with VLSI LISP (MEGACHIP)
- SYMBOLIC machines with the new VLSI ("IVORY")

For these products restricted access to military versions is not ensured by either of the manufacturers.

2.3 PROCEDURALIZATION

This approach is developed by the Central Research Laboratory of Thomson-Csf and applied in the group's "military" Divisions. Its asset is that is eliminated two troublesome points for airborne artificial intelligence applications:

- interfacing between symbolic and traditional languages
- the need for memory recovery in the computer.

The methodology is based on a high-level tool belonging to the SPOCK generic family which enables proceduralization of the expert systems.

ADVANTAGES:

- total memory management
- target language C or ADA: easy interfacing with the conventional portion of the application
- porting over to traditional computers.

RESERVATIONS:

- this tool is presently in the prototype stage.

3 THE SPOCK TOOL

Spock is the result of research at the "Expert System" laboratory of the Central Research Laboratory of Thomson-Csf conducted on the compiling of the basic knowledge to meet the needs of the military branches of Thomson-Csf for real time expert systems.

The compiling of rules appeared to be the key to obtaining high performance. THe approach followed enables total proceduralization of the rule language (first order logic) in the target language.

The RETE algorithm (basic principle of OPS-V, ART, and Knowledge-Craft) was expanded to totally compile the path of the filtering network in the code corresponding to the target language. Different target languages can be addressed: LISP and C are available, while LTR3 and ADA are currently being worked on.

SPOCK is:

An executive of expert systems satisfying the needs of real time applications by means of two services offered:

- control over memory allocation without the need for a "garbage collector"
- primitives for communication with the outside.

SPOCK is the heir to:

- OPS-V
    * Predominance of forward chaining in the rules
    * representation of objects with vectors
    * filtering with the RETE algorithm
- ART
    * philosophy of integrating several paradigms in a core of type OPS-V
- ML
    * prototyping language used for writing the rule compiler

SPOCK has some original points:

- the rules concern abstract objects which can be represented by any structure of the target language (defstruct or flavors in LISP, record in ADA, struct in C, etc.)
- total compiling of rules: the RETE network does not exist at run time, the path of the network has been compiled
- "garbage-free" running guaranteed so long as the rules do not explicitely invoke LISP
- rules provided with a digital priority in either static or dynamic form, cad calculated from the variables of the left member of the rule
- versatile representation of objects suitable for subsequent installation of hypothesis management mechanisms
- optimum compiling of the existential quantifier; possibility of resequencing the premises in the rules to optimize the tests performed upon running based on usage statistics
- invoking of a communication function after each inference cycle to take into account external events or to manage a schedule

4

SPOCK performs well:

A comparison of SPOCK with other tools with regard to the solving of a planning problem used by NASA to quantify the performance of such tools and the machines which support them ("Monkey and bananas," 30 rules) yielded the following results:

OUTIL (version) TOOL	MACHINE (mémoire) (memory)	Nb règles déclenchées Fired rules	Temps en secondes Time(s)	Nb règles par sec. Fired rules/s
SPOCK-C (V1)	SUN (RISC) 4-260	82	0.03	2733
SPOCK-C (V1)	SUN 3-160(4Mo)	82	0.14	586
SPOCK-Lisp (V1)	SUN 3-160(4Mo)	82	0.18	455
SPOCK-Flavors (V1)	SYMBOLICS 3620(8Mo)	82	0.31	259
ART (V2)	SYMBOLICS 3640(4Mo)	86	1.2	72
OPS-V (DEC V2)	VAX 11/780	81	1.3	62
OPS-V (Forgy V2)	SYMBOLICS 3640(4Mo)	81	1.7	48
ART (V2)	TI Explorer (4Mo)	86	2.4	36

Programming of rules for tools other than SPOCK was conducted by NASA.

Comments:

The performance comparison between SPOCK and OPS-V has two explanations:

- in OPS-V the filtering network is represented in the form of an interpreted microprogram, whereas for SPOCK the network path is totally compiled
- the processing of object deletions and modifications in SPOCK is an original variation of the RETE algorithm with, in addition, modification of objects "on location," that is, without copying, contrary to OPS-V.

These same reasons partially explainthe advantage of SPOCK over ART; another part is attributable to the lightening of certain processes in SPOCK compared to ART (for example, no uniqueness test for each insertion); on the other hand, certain operational acceleration solutions in ART (search in local memory through dynamic chopping on the equality tests, for example) are still not implemented in SPOCK.

The SPOCK-Lisp times depend on the representation of the objects to which the rules apply. With SYMBOLICS, better results could have been obtained by using "defstruct" rather than "flavors."

The similarity between the SPOCK-Lisp and SPOCK-C performance in this test is due to the absenve of arithmetic operations which, in LISP, would require type tests while running.

3.1 SPOCK INFERENCE CYCLE

SPOCK's inference cycle includes four phases:

- a filtering phase (1) (pattern matching) which, from the current set of objects (working memory) and the fixed set of rules, determines all the rules instantiated, that is, uplets? of the type: (rule, object1,..., objectN) formed by the name of a rule and a combination of objects which satisfies the left member of the rule.
- a selection phase (2) (conflict resolution) which determines the rule instance of highest priority; in the case of equal priorities the choice is random.
- a tripping phase (3) which consists in executing the right portion of the rule selected on the objects of the instance whose effect is to modify all the objects and/or to communicate actions to the outside world.

5

- a communication phase (4) which consists in executing the communication function; this function, which performs nothing by default, is to be redefined by the user, for example to subordinate a clock object to an external clock, take into account external modifications (data acquisition in real time), manage a schedule, etc.

Operation of the Inference Cycle:

SPOCK incrementally calculates all the rules instantiated in each inference cycle. Filtering phase 1 therefore corresponds to updating operations on all the conflicts which actually occur after each modification of the set of objects, that is, during phases 3 and 4.

In the initial state the set of objects and the set of rules instantiated are empty. Upon initialization several objects are inserted into the working memory, which can cause instantiation of several rules that are placed on the agenda. At the beginning of the inference cycle, the highest priority instance is triggered: these actions can possibly modify the working memory -- and consequently the agenda through filtering -- and the cycle is continued until one of the following conditions arises:

- during the selection stage the agenda is empty: the cycle stops
- during phases 3 or 4 the "halt" function is invoked: the cycle will stop during the following selection phase
- during any one of the phases, if an interruption occurs (break or error): the cycle stops immediately.

In the last case above resuming of the cycle is possible through the "restart" function provided that a coherent state is restored by invoking the "clearandmatch" function.

In the case of real time expert system applications, the inference cycle is to be included in a more general waiting or external-data-management loop which allows the inference cycle to be rebooted automatically.

3.2 OBJECTS MANAGED BY SPOCK

The objects to which pertain the rules are structures provided with abstract operations for creation, type testing, inspection, and assigning of attributes. In LISP the objects can be represented by any structure defined by "`defstruct`" or an extension by objects provided with active values, procedural attaching, multiple inheritance, etc., such as Flavors, for example.

The rules are generic and function independently of all the characteristics of the application objects.

In versions 2 and 3 of SPOCK the logical assertions can be represented by means of facts verifying a uniqueness test; the facts are internal objects which, contrary to abstract objects, cannot be shared with the rest of the application.

An example of objects is given under heading 4.2.2.

3.3 RULES MANAGED BY SPOCK

Their complexity varies according to the successive versions of SPOCK.

Version 1 of SPOCK has rules of the forward chaining type The rules have a name, a priority, a left portion, and a right portion.

The left portion expresses a condition on the working memory (in a logical type language of the first order).

The right portion expresses the actions to be taken when the left portion is true.

A rule is similar to a logical implication of the following type:

$$\mu\ o_1{:}t_1 \ldots \mu_k\ o_k{:}t_k \rightarrow \text{actions}$$

in which the objects o_i of type t_i are quantified by micron$_i$ universally, existentially, or negatively.

The types of objects are expressed by diagrams (patterns).

Examples of rules are given under heading 4.2.3.

Version 2 of SPOCK offers in addition a Truth Maintenance System and structuring of the rule data base in packs.

Version 3 will offer functions for structuring all the facts in a hierarchy of contexts and parallel exploring of several hypotheses.

3.3.1 LEFT MEMBERS OF THE RULES

The left member of a rule is a conjunction of conditions.

A condition can be:

- a diagram quantified universally, existentially, or negatively
- a call to LISP corresponding to a condition which is satisfied if the result is different from nil, and is false otherwise
- a link of a variable to the value of a LISP expression corresponding to a condition which is always satisfied (in order to store the result of an intermediate computation).

By default, the diagrams are universally quantified.

The existential quantification is identified syntactically by the key word `exists` in front of the diagram.

Negation is identified by the key work `not`.

For example:

```
ConditionsList :
                        condition
                        ConditionsList condition;
condition:              pattern
                        symbol':'pattern
                        EXISTS pattern
                        NOT pattern
                        STRING
                        symbol'='STRING;
```

A diagram expresses a type of object in the form of a conjunction of tests.

The first test concerns the structure to which the object belongs. This test is compiled with the LISP predicate `typep` which is true if the object is an instance of the structure or an extension of the structure.

The other tests pertain to the fields of objects or to the object as a whole.

The tests on a field follow the name of the field; they are composed of:

- either a predefined predicate and a value (constant, variable, or LISP expression)
- or a call to LISP in the form `'predicat arg2...argN` which is compiled as `(predicat arg1 arg2...argN)` where `arg1` is the value of the field.

The tests on the object as a whole are possible in the calls to LISP identified by the word `call` within the diagram. To do this all that need be done is to have the variable `self`, which is connected with the object being tried for the diagram, appear in the LISP expression

3.3.2 CONVENTION FOR THE ORDER OF THE TESTS

The rule compiler does not change the order of the premises in the rules nor the order of the tests in the diagrams.

The compiler divides all the network nodes that can be made common to several rules without disturbing the order of the tests.

3.3.3 THE RIGHT HAND MEMBERS OF THE RULES

The right hand members represent sequences of actions to be performed in the environment of the variables linked by the left members. An action can be:

- any LISP expression
- a link of intermediate variables to any values using bind
- a predefined action for modifying the working memory:
    * insertion of a new object
    * deletion of an object
    * modification of an object with and without filtering

3.3.4 PRIORITIES IN RULES

Priorities are whole or real numerical values.

The priority of a rule can be static or dynamic.

Dynamic priorities are defined by:

- the value of a linked variable in the left member
- any LISP expression that can pertain to the linked variables of the left member.

The dynamic priorities are evaluated during the filtering phase as soon as a rule instance is found.

The static priorities are defined by a whole or predefined constant (`maximum, minimum, immediate-firing`).

The `immediate-firing` constant indicates that the rule is triggered as soon as an instance is found. The rules using this priority must comply with some usage precautions and be reserved for simple actions.

4 APPLICATION OF SPOCK: TACTICAL ASSISTANT

The work in progress must first arrive at an assistance tool for reconaissance-type mission preparation using light helicopters, then later result in a navigation aid in a field of threats, taking into account the terrain by means of an airborne cartographic data base.

The expertise is concentrated in a Head Pilot of the Army Light Aviation (ALAT).

The problems to deal with are:

- taking into account of intervisibility between the helicopter and the possible threats (with better possible usage of the terrain)
- helicopter-target intervisibility (in the event of firing)
- generation of paths
- control of path generation taking into account the threats, terrain, and compliance with the mission (time, target).

In order to demonstrate the validity of SPOCK for this type of problem, a model simulator based on a toponymic description of a geographical zone was made on the SYMBOLICS machine.

4.1 SIMULATOR FOR THE TACTICAL ASSISTANT

4.1.1 INPUT PARAMETERS

The user defines:

- the start position of the helicopter
- the speed of the helicopter
- the final destination
- the various threats symbolized by:
    * their name
    * their position
    * their range
- the environment evolution rules
    * maintaining of of targets (generally reach the destination)
    * coherence of the decisions taken with the environment (avoid threats, for example).

4.2 SIMULATION METHOD

4.2.1 TAKING INTO ACCOUNT THE TIME ELEMENT

Simulation of time is obtained by management of a clock-type object which includes the Time information. The Time is incremented according to a minimum priority rule when all the objects in the system have been processed for the current Time instant.

The movement of the mobile objects is obtained in the same way by a minimum priority rule when the object is found to be late with respect to the Time variable. Thanks to this minimal priority the movement is not made until all the movement adjustment parameters (speed and direction) have been positioned.

4.2.2 STRUCTURE OF OBJECTS

The simulation implements four types of objects (Flavors):

- Helicopter
- Destination
- Obstacle(s)
- Clock

These objects are constructed from an inheritance of the following basic objects:

- TITLE: NAME field
- CLOCK: TIME field
- DIMENSION: RANGE field
- POSITION:
    * field X)
    * field Y) position in the horizontal plane
    * field Z altitude
- SPEED:
    * field Vx) variation in horizontal
    * field Vy) position for each movement
    * field Zz variation in altitude for each movement

The final objects are constructed by inheritance:

- OBJECT inherited from:
    * TITLE
    * POSITION
- MOBILITY inherited from:
    * SPEED
    * CLOCK
- MOBILE-OBJECT inherited from:
    * OBJECT
    * MOBILITY

- HELICOPTER inherited from MOBILE-OBJECT
- DESTINATION inherited from OBJECT
- OBSTACLE inherited from:
    - * OBJECT
    - * DIMENSION

NOTE: The real threats would be of the MOBILE-OBSTACLE type.

4.2.3 RULES FOR SITUATION EVOLUTION

The rules are formulated by the macrocommand DEFRULE which has as arguments:

- The Name of the rule (reusable by the plotting and program execution control tools)
- The Priority of the rule (used for the classification of the rule triggerings in the agenda)
- The Body of the rule: premises and then conclusions linked with the symbol "->"

Time incrementation rule:

This rule takes into account only one object: the CLOCK.

```
(DEFRULE INCREMENTATION 0
    H:=(HORLOGE TEMPS=Tps)
    →
    (MODIFY H TEMPS =(+1 Tps))
```

Object movement rule

```
(DEFRULE MOUVEMENT 0
    O:=(OBJET-MOBILE
            X=x
            Y=y
            Z=z
            Vx=vx
            Vy=vy
            Vz=vz
            TEMPS=tom)
   (HORLOGE
            TEMPS=Tps)
   (>Tps tom)   :l'objet est en retard sur le temps
   →
   (MODIFY O X=(+x vx)
             Y=(+y vy)
             Z=(+z vz)))
```

Rule for navigation toward target (destination)

In order to ensure that the helicopter is going to reach the target, its heading is compared with the bearing of the straight line joining its current position to that of the target; an excessively large difference will result in a heading modification to reduce the drift to a toleratable value.

```
(DEFRULE VERS-DESTINATION 3
     H:=(HELICOPTERE
              X=x
              Y=y
              Vx=vx
              Vy=vy)
     D:=(DESTINATION
              X=dx
              Y=dy)
     (>(-(ATAN vy vx)(ATAN (-dy y)(-dx x)) )0.1)
     →
     (MODIFY H X=(*(COS(ATAN (-dy y)(-dx x)))
                   (SQRT(+(* x x)(* y y)))
               Y=(*(SIN(ATAN (-dy y)(-dx x)))
                   (SQRT(+(* x x)(* y y)))))
```

Rule for avoiding obstacles (threats)

In order to avoid an obstacle (threat) we firstly consider those found along the helicopter's path; avoidance equals a change in heading.

The selection can be made according to two major criteria

- the helicopter is located within the range of the threat
- the helicopter will be within the range of the threat in k units of Time if it does not change its heading (or speed or altitude).

The first criteria corresponds to the appearance of a threat either because the threat is mobile or because a change in the terrain has led to an intervisibility between the helicopter and the threat; this criteria is not yet taken into account in the simulator which uses the second criteria with k=3.

10

Various path correction strategies can be implemented; it was decided that the direction of the tangent to the threat's range circle would be taken as the new heading at the point where the path of the helicopter would have crossed the threat's range circle.

```
(DEFRULE EVITE-OBSTACLE 2
    H:=(HELICOPTERE
            X=x
            Y=y
            Vx=vx
            Vy=vy)
    O:=(OBSTACLE
            X=ox
            Y=oy
            PORTEE=p)
    (<=(+(EXPT (- ox (+ x(* 2 vx)))2)
          (EXPT (- oy (+ y(* 2 vy)))2))
        (EXPT p 2))
    →
    (MODIFY H
        Vx=(* (SQRT (+(EXPT vx 2)(EXPT vy 2)))
               (COS (+(ATAN (-oy y)(-ox x))

                      (*(SIGNUM(-(ATAN (-oy y)(-ox x))
                                  (ATAN (vy vx)))
                        (/ PI -2)))))
        Vy=(* (SQRT (+(EXPT vx 2)(EXPT vy 2)))
               (SIN (+(ATAN (-oy y)(-ox x))
                      (*(SIGNUM(-(ATAN (-oy y)(-ox x))
                                  (ATAN (vy vx)))
                        (/ PI -2)))))))
```

4.3 PERFORMANCE

For a limited number of obstacles (less than 30) we obtain a performance between 200 and 250 rules triggered per second, which is perfectly coherent with the results obtained with the NASA benchmark (cf. Table).

4.4 PLANNED IMPROVEMENTS

The simulator portion will take into account the notion of volume of threat and will be linked to the cartographic data base equipment in order to take into account the terrain both for navigation as well as for avoiding threats.

REFERENCES

[GD 87]

M. Ghallab, P. Dufresne -- Inference motors for systems of production rules: compiling and interpretation techniques. LAAS Toulouse, 1987.

[All other references already in English]

NASA

Report Documentation Page

1. Report No. TT-20357	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle SPOCK--A TOOL FOR REAL TIME ARTIFICIAL INTELLIGENCE		5. Report Date AUGUST 1988
		6. Performing Organization Code
7. Author(s) J. P. LaCroix, G. A. Berthon, F. Fages, P. Repusseau		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address National Aeronautics and Space Administration Washington, DC 20546		11. Contract or Grant No. NASW-4307
		13. Type of Report and Period Covered translation
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION WASHINGTON, DC 20546		14. Sponsoring Agency Code

ORIGINAL PAGE IS OF POOR QUALITY

15. Supplementary Notes

Transl. from FRENCH to ENGLISH of "Spock, un Outil pour l'Intelligence Artificielle Temps Reel", AGARD, 55th Meeting of the Avionics Panel of Software and Engineering and its Application to Avionics. Held in Cesme, Turkey, Apr. 25-29, 1988
Transl. by SCITRAN, Santa Barbara, CA 93150.

16. Abstract

Airborne expert systems must take into account very harsh real time conditions which, when combined with the environmental conditions of the object computer, require the development of new solutions. The proceduralization obtained by compiling the RETE algorithm toward symbolic, procedural, or real-time oriented target languages provides a solution to the primary problems of airborne applications: speed requirements, memory control, harmonization of symbolic parts with the traditional portions of the applications. This approach is supported by the SPOCK family of tools (-LISP, -C,LTR3, -ADA) developed by the Central Research Laboratory of Thomson-Csf which offers significant improvements over presently available products. One of the SPOCK applications in Thomson-Csf is a helicopter mission aid system which takes into consideration the threats and terrain thanks to integration with an airborne cartographic data base.

17. Key Words (Suggested by Author(s)) [illegible]	18. Distribution Statement unclassified-unlimited

19. Security Classif. (of this report) unclassified	20. Security Classif. (of this page) unclassified	21. No. of pages 17	22. Price